Maintenance Information for Remote Commanding

5 October 2006

1 Remote commanding maintenance documentation

This document is intended to aid programmers who maintain and enhance the remote commanding applications. It helps to be familiar with the user's guide before reading this document.

Source code for all applications is in src/javadisplay/remcmd.

Javadoc

Javadoc html files are a good source of information on these applications. Unfortunately, javadoc in java 1.2 must be used to get useful files and as of this writing java1.2 is not installed on all of our machines. The script '`runjavadoc`' can be used to generate javadoc files, but you may have to modify the path it uses to locate javadoc 1.2.

package remcmd

All source files for these applications declare that they are part of the "remcmd" package. As a result:

- Class files can be executed only when they're in a directory named "remcmd". Make install copies class files to classes/remcmd
- $CLASSPATH must contain the parent of the remcmd directory.
- The package name must be used when invoking an application, for example `java remcmd.RecvCmd`.
- When a source file inside the package references a class or interface outside the package, an import statement is required. Only public methods and members can be referenced.

Application modules
Inter-application message formats
Threads
Creating tar files
Installation details
RelayCmd information

2 Application Modules

These applications were developed in response to a request from the Ultra Long Distance Balloon mission, but it seems likely that other missions will be interested in them in the future. It is inevitable that those missions will want different functionality from what ULDB wanted. To deal with this need for flexibility, parts of the applications that seem most likely to change have been isolated into modules and an inteface has been defined for each. At runtime, the name of the class that will implement each interface can be specified in a command-line argument.

The following module interfaces have been defined:

- **Encryptor** - used by SendCmd and RecvCmd
  This interface is currently implemented by classes GPG1, which uses gpg for encryption, and NoEncryption, which doesn't encrypt data. NoEncryption might be used when both SendCmd and RecvCmd are on the same secure network. Other classes implementing of this interface might use some encryption package other than gpg. GPG1 is the default for both SendCmd and RecvCmd.
- **Decryptor** - used by SendCmd and RecvCmd
  This interface is implemented by classes GPG1 and NoDecryption. GPG1 is the default for both SendCmd and RecvCmd.
- **Log** - used by all three applications to write log files.
  This interface is implemented by classes Log1, which lets the user set the log file name and directory, and NoLog, which does not create a log file. Log1 is the default for SendCmd and RecvCmd. NoLog is the default for RelayCmd.
- **CmdBuilder** - used in SendCmd to let users create STOL commands.
  Two classes currently implement this interface.
  1. TextFieldBuilder presents the user with a text input field into which the command is typed.
  2. MenuBuilder extends TextFieldBuilder and displays a menu with a button for each command. After the command is built, the TextFieldBuilder frame is displayed so the user can check the final form of the command, add comments or change the default expiration interval.
- **RecvComm** - used in SendCmd to communicate with RecvCmd.
  RecvComm1 is the only class to implement this interface at the moment. It communicates with RecvCmd over a TCP/IP socket. Other implementations might user datagrams or mail. The class used by SendCmd as its RecvComm module must be compatible with the class used by RecvCmd as its SendComm module.
- **SendCmdDB** - used in SendCmd to maintain a database of commands that have been sent recently.
  This interface is currently implemented by NoSendDB, which does not maintain any database, and SendDB1. The benefit of maintaining a database is that recent commands can be displayed, allowing the user to examine all messages concerning a command and to ping RecvCmd for a command's status. SendDB1 is the default.
- **SendComm** - used in RecvCmd to communicate with SendCmd.
  SendComm1 is the only class currently implementing this interface. It uses TCP/IP

sockets to communicate with SendCmd, and is compatible with RecvComm1, used by SendCmd.

- **ItosComm** - used in RecvCmd to pass commands STOL.
  ItosComm1 is the only class currently implementing this interface. It passes commands to STOL over the stol fifo.
- **RecvCmdDB** - used in RecvCmd to maintain the database of commands received.
  This interface is implemented by classes NoRecvDB, which does not maintain any database, and RecvDB1. RecvDB1 maintains a record of all actions concerning each command and determines when a command expires and can no longer be accepted.
- **Acceptor** - used in RecvCmd to allow the user to accept or reject commands.
  This interface is implemented by classes AcceptAll and Acceptor1. AcceptAll immediately accepts all commands received and passes them to STOL. Accpetor1 displays a list of commands received and does not pass a command to STOL until the user explicitly accepts it.
- **ClosedItosComm** - used in RelayCmd to communicate with RecvCmd or another RelayCmd
  This interface is implemented by class RelayComm1, which can be either server or host in the connection. It uses a TCP/IP socket that is kept open even when no data is being exchanged.

3 Inter-application message formats

This chapter describes the format of messages sent between applications. At the moment, these messages are sent over TCP/IP sockets.

3.1 Fields present in all messages

All messages start with six digit characters. That integer gives the length of the remainder of the message. These six characters are never encrypted, but when encryption is used everything following them is encrypted.

After decryption, all fields following the six-digit length field are terminated by line breaks. Encryption inserts canonical line breaks, so when the message is decrypted, breaks appropriate for the receiving system are inserted.

The first two fields following the message length are always:

- The version number of the sending application in floating-point format. This field is ignored at the moment but might be used in the future after a change in message format.
- An integer giving the message type. Four types are currently defined: SendCmd sends COMMAND_MSG and PING_MSG. RecvCmd sends PING_REPLY and COMMAND_INFO_MSG. Integer constants for those four types are defined in RemcmdUtil.java

3.2 COMMAND_MSG message format

SendCmd sends STOL commands to RecvCmd in COMMAND_MSG messages. Fields following the message type field are:

1. The IP address to which replies to this command will be sent. That is normally the host where SendCmd is running.
2. The port number to which replies are sent.
3. The name of the gpg user whose public key should be used to encrypt replies to this command.
4. The text of the STOL command.
5. An ID string for the command.
6. An integer giving the expiration interval for the command in minutes, or one of the constants Cmd.NO_EXPIRATION or Cmd.DEFAULT_EXPIRATION.
7. The name of the sender. This may be the empty string.
8. An integer giving the command's current status. In COMMAND_MSG, this is usually ENCRYPTED. Status constants are defined in Cmd.java. This field is not useful in COMMAND_MSG messages but is included so that command descriptions in all messages contain the same fields. This allows a constructor for class Cmd to create a Cmd object from any message.
9. An optional, multi-line comment.

10. The end-of-command string defined in RemcmdUtil.endCmdString

Fields 4 through 9 are created by Cmd.toMsg(). The remaining fields are created by SendCmd.processCmd();

3.3 PING_MSG message format

SendCmd sends a PING_MSG when the user clicks the "Check connection to ITOS" or "Ping this command" buttons. Fields following the message type field are:

1. The IP address to which replies to this ping will be sent. That is normally the host where SendCmd is running.
2. The port number to which replies are sent.
3. The name of the gpg user whose public key should be used to encrypt replies to this ping.
4. Text of the command being pinged. If This ping was initiated by the "Check connection to ITOS" button, this field is empty.
5. The ID string for the command being pinged. If this ping was initiated by the "Check connection to ITOS" command, this field contains the reserved string Ping.CONNECTION.

   Ping defines two other reserved strings, SENDER and HOST. SENDER would request information on all commands sent by a particular person, regardless of the host. HOST would request information on all commands sent from the host. These functions have not been implemented but could be if they were requested.
6. The name of the sender. This may be the empty string.

Fields 4 through 6 are created by Ping.toMsg(). The remainder are created by SendCmd.processPing().

3.4 PING_REPLY message format

RecvCmd sends a PING_REPLY message in response to every PING_MSG it receives. Fields following the message type are:

1. An integer giving the number of commands described in this message. At the moment this field is always "1". It will be greater than one if pings are implemented that request information on all commands sent from a host or by a user.
2. For each command, fields 4 through 10 described above for COMMAND_MSG are inserted.

   When the ping asked about the connection to ITOS, the command text and ID (fields 4 and 5) are set to Ping.CONNECTION. A string describing the connection to ITOS is put into the comment field (field 9).

3.5 COMMAND_INFO_MSG

RecvCmd sends a COMMAND_INFO_MSG when the operator sends a reply to the sender and when there is an important change in a command's status, such as when it is accepted or rejected.

Fields following the message type are identical to those in a PING_REPLY message. If the operator is sending a reply, the text of the reply is put into the command's comment field (field 9).

4 Threads in remcmd applications

SendCmd threads

RecvThread
: SendCmd and RecvCmd open a socket connection each time one has a message to send and close the socket immediately. RecvComm1, the module in SendCmd that takes care of communications with RecvComm, starts a single RecvThread object that waits for connections on a server socket, reads one message over the socket and closes the client socket. The message is put into the decryption/encryption job queue.

SendDBThread
: SendDB1, a database module for SendCmd, starts a single SendDBThread that occasionally looks through the database for commands that are past their deletion times. If any are found, they are deleted from the database and the database is saved to disk. This search is done every five minutes at present.

RecvCmd threads

FifoWriterThread
: ItosComm1, the module in RecvCmd that passes commands to STOL, starts a FifoWriterThread each time it wants to send a command to STOL. The thread opens a FileOutputStream, writes a command to the stream and exits. This is done because if the stol fifo exists but STOL is not running, the constructor of a FileOutputStream to write to the fifo hangs indefinitely.

RecvDBThread
: RecvDB1, a database module for RecvCmd, starts a RecvDBThread that occasionally looks through the command database for commands that have expired (can no longer be accepted) or that should be deleted.

SendThread
: SendComm1, the module that communicates with SendCmd, starts a single SendThread object that waits for connections from SendCmd clients. Each time a client connects, the thread reads one message, closes the client socket and puts the message into the job queue.

GPG1 threads

GPG1, the gpg encryption/decryption module used by both SendCmd and RecvCmd creates these threads:

JobThread
: Under some circumstances, encryption and decryption can take a substantial amount of time, so they are done asynchronously. When GPG1 encrypts or

decrypts text, it does so by putting a request into the job queue. One JobThread object is started, which waits for jobs to be put into the queue and executes them.

CatchOutput
: GPG1.runGpg() starts a child process to run all GPG commands. Each time it does so, it starts a CatchOutput thread to read anything GPG writes to stderr and put it into a String. The thread exits when stderr is closed. The code allows another thread to order a CatchOutput thread to exit prematurely, but I don't think this ability is being used.

RelayCmd threads

ServerThread
: RelayComm1 is the module in RelayCmd that communicates with RecvCmd or with another RelayCmd. If RelayCmd will be the server on this connection, RelayComm1 starts a ServerThread that opens a socket and waits for the client to connect. ServerThread then reads all messages that come over the socket and puts them into the job queue. If the socket connection is broken, the thread goes back to waiting for the client to connect.

ClientThread
: If RelayCmd will be the client on this connection, RelayComm1 starts a ClientThread that tries to connect to the server every 10 seconds. After connecting, the thread reads all messages from the server and puts them into the jobqueue. If the socket connection is broken, the thread goes back to trying to connect every 10 seconds.

5 Creating tar files

The following paragraphs are only here for historical documentation purposes. The shell script 'maketar' no longer has to manually run because it is now part of the "rercmd" make file. This make file builds the tar files listed below in the '$TCWDIR/pkgs' directory which is just below the ITOS root directory and is created when "make install" is run.

The shell script 'maketar' in '$ITOS_DIR/src/javadisplay/remcmd' can be used to create tar files. It accepts the following options:

`gpgdir <path>`

Names the directory where gpg executables are stored. Executables should be named gpg.<os>, where <os> describes the executable's operating system. <os> will be used to create the tarfile name. Example names:

- gpg.i386-FreeBSD-3.2
- gpg.i386-SunOS-5.6
- gpg.sparc-SunOS-5.7

Default is '$ITOS_DIR/src/javadisplay/remcmd/gpgbin', where several executables have been committed to cvs.

`classes <path>`

Names a directory containing subdirectory remcmd, which contains the remote commanding class files. Default is '$ITOS_DIR/classes'

`sxdir <path>`

Names the directory containing the sx class files. They are used to parse and check strings entered as submnemonic values. Default is '$ITOS_DIR/src/javadisplay/sx'

`scripts <path>`

Names the directory containing scripts and other files that go into the tar files. Default is '$ITOS_DIR/src/javadisplay/remcmd'

`tempdir <path>`

Names the directory where files are put during construction of tarfiles. If directory <path> currently exists, it is emptied. Default is './tar_temp'

`tardir <path>`

Names the directory where tar files generated by this script are put. Default is the current directory.

Note: All directories must be entered as absolute paths, not paths rooted at the current directory or beginning with '~'.

One tarfile is made for each gpg executable in gpgdir and an additional tarfile is made without a gpg executable.

So a set of tarfiles could be made like this:

```
cd $ITOS_DIR/src/javadisplay/remcmd
maketar tardir $ITOS_DIR/src/javadisplay/remcmd/tars
```

In this example, 'maketar' could be run without the `tardir` option, but then the tar files would be put into remcmd along with all the .java, .class and other files.

6 Installation details

These applications attempt to be as flexible as possible. They don't assume where gpg is installed or where its key files are stored. They don't make any assumptions about where the class files reside or where the run scripts or argument files reside. They don't assume that argument files will be used.

The INSTALL script is provided for users who don't care where things are put and just want to get started with a minimum of hassle. INSTALL assumes that files are left in the directories they have in the tar files. That is,

- Runscripts, argument files and INSTALL are all in one directory.
- The gpg executable and startup key files are in a subdirectory of that directory, named '`gpg`'.
- The sx class files and the gnu jar file are in another subdirectory of that directory named '`classes`'.
- Remcmd class files are in a subdirectory of '`classes`' named '`remcmd`'.

INSTALL does the following:

- Edits all run scripts by setting $argDir to the current directory, meaning the argument files are in the same directory as the run scripts.
- Edits the rungpg script by setting $gpgDir to the gpg subdirectory.
- Edits sendargs and recvargs by setting both gpgpath and keydir to the gpg subdirectory.

At the moment, the source files for remote commanding are included in ITOS releases and "make install" creates the class files and puts them into classes/remcmd. So strictly speaking, no remcmd tar file is needed to use remote commanding on a machine where ITOS has been installed.

However, setting up argument files and shell scripts with the INSTALL script is much easier than doing it by hand, and that script works only when files are in the directory structure contained in the tar files, so you may want to install remote commanding from a tar file even on ITOS machines.

7 RelayCmd

Some parts of RelayCmd are less clear than they might be, and this chapter tries to explain them.

RelayCmd receives messages from two sides, one side toward RecvCmd, the other toward SendCmd applications. On both sides, there may be one or more additional instances of RelayCmd before the other applications. RelayCmd's job is simply to forward messages received on one side to the application on the other side. RelayCmd does not decrypt messages.

A complication is that some information must be passed with these messages that is not normally sent between applications. For example, the RelayCmd that gets a message from a SendCmd knows SendCmd's IP address. That must be sent along with the message to RecvCmd. (The message contains an IP address, but it is sometimes "localhost", not very useful.) Similarly, when RecvCmd decrypts the message, it finds the port on which SendCmd is waiting for a reply. That must be sent with the message back to the RelayCmd that will finally send it to SendCmd.

As a consequence, messages going in the two directions have slightly different structures. Messages going toward RecvCmd are received in IncomingMsg objects, which contain two members, the IP address of the SendCmd where the message originated and an encrypted block of text. Messages going toward SendCmd are received in IncomingMsg2 objects, which have a third field, the port number where the SendCmd application is waiting for it.

Class RelayComm1 is used at both ends of all sockets between RecvCmd and RelayCmd and between two RelayCmd applications. To do that, it implements interfaces SendComm and ClosedItosComm. When it sends an IncomingMsg or IncomingMsg2 over a socket it uses RemcmdUtil.sendCryptText(), the method used by all remcmd applications. That method always sends six decimal characters and then a block whose length is given by the six characters. RemcmdUtil.sendCryptText() is called twice to send an IncomingMsg or IncomingMsg2. The block in the first call contains a normal application message. The block in the second message contains a host address and optionally a port number.